Method and Apparatus for Allocating and De-allocating Consecutive Blocks of Memory in Background Memory Management

by inventors Enrique Musoll and Mario Nemirovsky

5

Cross-Reference to Related Documents

10   The conception of the invention was documented in Document Disclosure No. 491,557, entitled *Hardware Algorithm for Allocating and De-allocationg Consecutive Blocks of Memory*", filed on 04/03/01. The present invention is a continuation in part (CIP) to a U.S. patent application S/N 09/602,279 entitled *"Methods and Apparatus for Background Memory Management"* filed on 06/23/00. The present invention is also a

15   CIP to a U.S. patent application S/N 09/737,375 entitled *"Queuing System for Processors in Packet Routing Operations"* and filed on 12/14/00, the latter claiming priority to a provisional patent application S/N 60/181,364 filed on 02/08/00. The referenced applications are included herein by reference.

20

Field of the Invention

        The present invention is in the area of integrated circuit

25   microprocessors, and pertains in particular to memory management, and the use of microprocessor resources in such management.

30

Background of the Invention

Microprocessors, as is well-known in the art, are integrated circuit (IC) devices that are enabled to execute code sequences which may be generalized as software. In the execution most microprocessors are capable of both logic and arithmetic operations, and typically modern microprocessors have on-chip resources (functional units) for such processing.

Microprocessors in their execution of software strings typically operate on data that is stored in memory. This data needs to be brought into the memory before the processing is done, and sometimes needs to be sent out to a device that needs it after its processing.

There are in the state-of-the-art two well-known mechanisms to bring data into the memory and send it out to a device when necessary. One mechanism is loading and storing the data through a sequence of Input/Output (I/O) instructions. The other is through a direct-memory access device (DMA).

In the case of a sequence of *I/O* instructions, the processor spends significant resources in explicitly moving data in and out of the memory. In the case of a DMA system, the processor programs an external hardware circuitry to perform the data transferring. The DMA circuitry performs all of the required memory accesses to perform the data transfer to and from the memory, and sends an acknowledgement to the processor when the transfer is completed.

In both cases of memory management in the art the processor has to explicitly perform the management of the memory, that is, to decide whether the desired data structure fits into the available memory space or does not, and where in the memory to store the data. To make such decisions the processor needs to keep track of the regions of memory

wherein useful data is stored, and regions that are free (available for data storage). Once that data is processed, and sent out to another device or location, the region of memory formerly associated with the data is free to be used again by new data to be brought into memory. If a data structure fits into the available memory, the processor needs to decide where the data structure will be stored. Also, depending on the requirements of the processing, the data structure can be stored either consecutively, in which case the data structure must occupy one of the empty regions of memory; or non-consecutively, wherein the data structure may be partitioned into pieces, and the pieces are then stored into two or more empty regions of memory.

An advantage of consecutively storing a data structure into memory is that the accessing of this data becomes easier, since only a pointer to the beginning of the data is needed to access all the data.

When data is not consecutively stored into the memory, access to the data becomes more difficult because the processor needs to determine the explicit locations of the specific bytes it needs. This can be done either in software (i.e. the processor will spend its resources to do this task) or in hardware (using a special circuitry). A drawback of consecutively storing the data into memory is that memory fragmentation occurs. Memory fragmentation happens when the available chunks of memory are smaller than the data structure that needs to be stored, but the addition of the space of the available chunks is larger than the space needed by the data structure. Thus, even though enough space exists in the memory to store the data structure, it cannot be consecutively stored. This drawback does not exist if the data structure is allowed to be non-consecutively stored.

Still, a smart mechanism is needed to generate the lowest number of small regions, since the larger the number of small regions that are used by a

data structure, the more complex the access to the data becomes (more specific regions need to be tracked) regardless of whether the access is managed in software or hardware as explained above.

A background memory manager (BMM) for managing a memory in a data processing system is known to the inventor. The memory manager has circuitry for transferring data to and from an outside device and to and from a memory, a memory state map associated with the memory, and a communication link to a processor. The BMM manages the memory, determining if each data structure fits into the memory, deciding exactly where to place the data structure in memory, performing all data transfers between the outside device and the memory, maintaining the memory state map according to memory transactions made, and informing the processor of new data and its location. In preferred embodiments the BMM, in the process of storing data structures into the memory provides an identifier for each structure to the processor. The system is particularly applicable to Internet packet processing in packet routers.

Because software-managed memory is costly in terms of developing instructions to figure out which portions of memory within a memory block are free and which are available, a hardware mechanism such as the one described with reference to S/N 09/602,279 enables more efficiency and therefore, cost savings. However, in order to optimize the function of such a hardware controller, a process must be provided to enable integrated and optimum function between hardware control and software control of memory. One of the preferred areas of use for such innovation is in the area of packet processing in data routing over networks.

What is clearly needed is a protocol that enables low fragmented packet queuing and de-queuing using on-board memory and hardware, wherein the memory is controlled in a manner to alleviate management responsibility traditionally assigned to CPU and other processor resources.

Summary of the Invention

In a preferred embodiment of the present invention a system for allocating storage of incoming data packets into a memory of a packet processor is provided, comprising a first facility mapping a first block of memory of a fixed block size in bytes into an ordered plurality of atomic pages comprising each a fixed byte size, a second facility mapping the same block of memory into ordered virtual pages of different sizes, ranging from a smaller virtual page size equal to the atomic page size up to a larger virtual page size equal to the fixed block size, a third facility to allocate virtual pages as unavailable for storage or de-allocate virtual pages as available for storage, a fourth facility to receive a data packet, ascertain packet size for the received packet, and to determine fit by checking allocation state for virtual pages of a smallest size that is equal to or larger than the packet size, then allocation state for next larger virtual pages, and so on, until a de-allocated, available virtual page is found; and a fifth facility to select a virtual page to store the packet, and to update and mark associated atomic pages in the selected virtual page as available or unavailable for storage, in an ordered manner. After each selection state of all atomic and virtual pages is updated.

In some preferred embodiments the system implemented in hardware. Also in preferred embodiments the second facility maps virtual pages in ascending orders of two from the atomic page size up to the block size. In a particular embodiment the block size is 64 KiloBytes (KB), mapped into 256 atomic pages of 256 Bytes each, and further mapped into 256 virtual pages of 256 bytes each, 128 virtual pages of 512 bytes each, and ascending in powers of two to two virtual pages of 32 KB each, and one virtual page of 64 KB.

In some embodiments there is further a mechanism for enabling groups of virtual pages by size, wherein the fifth facility selects only among enabled groups of virtual pages. In this system the fifth facility selects a de-allocated virtual page for storing the packet only from the enabled virtual page group of the smallest size that is still equal to or larger than the packet size.

In some embodiments there is a second block of memory of the same fixed block size as the first block of memory, mapped in the same way as the first block of memory, wherein a block is selected for storage based on state of enabled virtual page groups in each block, and then a virtual page is selected in the selected block based on fit. In some cases groups of virtual pages are mapped as enabled in an ascending order from a first block to a last block, having the effect of reserving lower-order blocks for smaller packet size.

In another aspect of the invention a data packet router is provided, comprising external ports to receive and send data packets from and to neighboring connected routers, and a packet processor having an on-board memory, and comprising a system for allocating storage of data packets in the on-board memory, the system having a first facility mapping a first block of memory of a fixed block size in bytes into an ordered plurality of atomic pages comprising each a fixed byte size, a second facility mapping the same block of memory into ordered virtual pages of different sizes, ranging from a smaller virtual page size equal to the atomic page size up to a larger virtual page size equal to the fixed block size, a third facility to allocate virtual pages as unavailable for storage or de-allocate virtual pages as available for storage, a fourth facility to receive a data packet, ascertain packet size for the received packet, and to determine fit by checking allocation state for virtual pages of a smallest size that is equal to or larger than the packet size, then allocation state for next larger virtual pages, and

so on, until a de-allocated, available virtual page is found, and a fifth facility to select a virtual page to store the packet, and to update and mark associated atomic pages in the selected virtual page in an ordered manner. In preferred embodiments the system for storing is implemented in hardware.

In some embodiments of the router, after each selection by the fifth facility, state of all atomic and virtual pages is updated. Also in some embodiments the second facility maps virtual pages in ascending orders of two from the atomic page size up to the block size. In some preferred embodiments the block size is 64 KiloBytes (KB), mapped into 256 atomic pages of 256 Bytes each, and further mapped into 256 virtual pages of 256 bytes each, 128 virtual pages of 512 bytes each, and ascending in powers of two to two virtual pages of 32 KB each, and one virtual page of 64 KB.

In some cases the hardware system further comprises a mechanism for enabling groups of virtual pages by size, and wherein the fifth facility selects only among enabled groups of virtual pages. In some of these embodiments the hardware system the fifth facility selects a de-allocated virtual page for storing the packet only from the enabled virtual page group of the smallest size that is still equal to or larger than the packet size.

In some preferred embodiments the on-board memory further comprises a second block of memory of the same fixed block size as the first block of memory, mapped in the same way as the first block of memory, wherein the hardware system selects a block for storage based on state of enabled virtual page groups in each block, and then a virtual page is selected in the selected block based on fit.

In another aspect of the invention a method for allocating storage for data packets in a memory of a packet processor is provided, comprising the steps of (a) mapping, by a first facility, a first block of memory of a fixed block size in bytes into an ordered plurality of atomic pages comprising

each a fixed byte size, (b) mapping, by a second facility, the same block of memory into ordered virtual pages of different sizes, ranging from a smaller virtual page size equal to the atomic page size up to a larger virtual page size equal to the fixed block size, (c) allocating, by a third facility, virtual

5   pages as unavailable for storage or de-allocating virtual pages as available for storage, (d) receiving a data packet by a fourth facility, ascertaining packet size for the received packet, and determining fit by checking allocation state for virtual pages of a smallest size that is equal to or larger than the packet size, then allocation state for next larger virtual pages, and

10   so on, until a de-allocated, available virtual page is found, and (e) selecting a virtual page by a fifth facility, to store the packet, and updating and marking associated atomic pages in the selected virtual page in an ordered manner.

In some embodiments of the method, in step (b), the second facility

15   maps virtual pages in ascending orders of two from the atomic page size up to the block size. Also in some embodiments the block size is 64 KiloBytes (KB), mapped into 256 atomic pages of 256 Bytes each, and further mapped into 256 virtual pages of 256 bytes each, 128 virtual pages of 512 bytes each, and ascending in powers of two to two virtual pages of 32 KB each,

20   and one virtual page of 64 KB.

In some embodiments there is a mechanism for enabling groups of virtual pages by size, wherein the fifth facility selects only among enabled groups of virtual pages. In some cases the fifth facility selects a de-allocated virtual page for storing the packet only from the enabled virtual

25   page group of the smallest size that is still equal to or larger than the packet size.

In some embodiments of the method there is a second block of memory of the same fixed block size as the first block of memory, mapped in the same way as the first block of memory, wherein a block is selected for

storage based on state of enabled virtual page groups in each block, and then a virtual page is selected in the selected block based on fit. In some cases enabled groups of virtual pages are mapped as enabled in an ascending order from a first block to a last block, having the effect of reserving lower-order blocks for smaller packet size.

In various embodiments of the invention taught in enabling description below, for the first time a hardware mechanism and a method is provided for selecting storage location in an on-board memory of a packet processor, wherein fragmentation is held at a minimum, and packets of various sizes may be forced into specific blocks.

Brief Description of the Drawing Figures

Fig. 1 is a simplified diagram of memory management by direct I/O processing in the prior art.

Fig. 2 is a simplified diagram of memory management by direct memory access in the prior art.

Fig. 3 is a diagram of memory management by a Background Memory Manager in a preferred embodiment of the present invention.

Fig. 4 is a block-diagram illustrating a hardware-controlled memory portion of a total processor memory.

Fig. 5 is a block-diagram illustrating layout of virtual pages for a division of the hardware-controlled memory of Fig. 4 according to an embodiment of the present invention.

Fig. 6a is a block-diagram illustrating a Fits Determination logic according to an embodiment of the present invention.

Fig. 6b is a block diagram illustrating an allocation matrix according to an embodiment of the present invention.

Figs 7a through 8d are block-diagrams illustrating a sequence of packet storage involving a plurality of different sized data packets according to an embodiment of the present invention.

Fig. 9 is a block diagram illustrating a comparison between

5   consecutive and non-consecutive data storage.

Description of the Preferred Embodiments

10   Fig. 1 is a simplified diagram of memory management in a system 104 comprising a processor 100 and a memory 102 in communication with a device 106. In this example it is necessary to bring data from device 106 into memory 102 for processing, and sometimes to transmit processed data from memory 102 to device 106, if necessary. Management in this prior art

15   example is by processor 100, which sends I/O commands to and receives responses and/or interrupts from device 106 via path 108 to manage movement of data between device 106 and memory 102 by path 110. The processor has to determine whether a data structure can fit into available space in memory, and has to decide where in the memory to store incoming

20   data structures. Processor 100 has to fully map and track memory blocks into and out of memory 102, and retrieves data for processing and stores results, when necessary, back to memory 102 via path 114. This memory management by I/O commands is very slow and cumbersome and uses processor resources quite liberally.

25   Fig. 2 is a simplified diagram of a processor system 200 in the prior art comprising a processor 100, a memory 102 and a direct memory access (DMA) device 202. This is the second of two systems by which data, in the conventional art, is brought into a system, processed, and sent out again, the first of which is by I/O operations as described just above. System 200

comprises a DMA device 202 which has built-in intelligence, which may be programmed by processor 100, for managing data transfers to and from memory 102. DMA device 202 is capable of compatible communication with external device 106, and of moving blocks of data between device 102

5   and 106, bi-directionally. The actual data transfers are handled by DMA device 202 transparently to processor 100, but processor 100 must still perform the memory mapping tasks, to know which regions of memory are occupied with data that must not be corrupted, and which regions are free to be occupied (overwritten) by new data.

10   In the system of Fig. 2 DMA processor 100 programs DMA device 202. This control communication takes place over path 204. DMA device 202 retrieves and transmits data to and from device 106 by path 208, and handles data transfers between memory 102 and processor 100 over paths 204 and 206.

15   In these descriptions of prior art the skilled artisan will recognize that paths 204, 206 and 208 are virtual representations, and that actual data transmission may be by various physical means known in the art, such as by parallel and serial bus structures operated by bus managers and the like, the bus structures interconnecting the elements and devices shown.

20   Fig. 3 is a schematic diagram of a system 300 including a Background Memory Manager (BMM) 302 according to an embodiment of the present invention. BMM 302 a hardware mechanism enabled to manage the memory in the background, i.e. with no intervention of the processor to decide where the data structure will be stored in the memory. Thus, the

25   processor can utilize its resources for tasks other than to manage the memory.

The present invention in several embodiments is applicable in a general way to many computing process and apparatus. For example, in a preferred embodiment the invention is applicable and advantageous in the

processing of data packets at network nodes, such as in routers in packet routers in the Internet. The packet processing example is used below as a specific example of practice of the present invention to specifically describe apparatus, connectivity and functionality.

5   In the embodiment of a packet router, device 106 represents input/output apparatus and temporary storage of packets received from and transmitted on a network over path 308. The network in one preferred embodiment is the well-known Internet network. Packets received from the Internet in this example are retrieved from device 106 by BMM 302, which

10   also determines whether packets can fit into available regions in memory and exactly where to store each packet, and stores the packets in memory 102, where they are available to processor 100 for processing. Processor 100 places results of processing back in memory 102, where the processed packets are retrieved, if necessary, by BMM on path 312 and sent back out

15   through device 106.

In the embodiment of Fig. 3 BMM 302 comprises a DMA 202 and also a memory state map 304. BMM 302 also comprises an interrupt handler in a preferred embodiment, and device 106 interrupts BMM 302 when a packet is received. When a packet is received, using DMA 202 and

20   state map 304, the BMM performs the following tasks:

1. Decides whether a data structure fits into the memory. Whether the structure fits into memory, then, is a function of the size of the data packet and the present state of map 304, which indicates those regions of memory

25   102 that are available for new data to be stored.

2. If the incoming packet in step 1 above fits into memory, the BMM determines an optimal storage position. It was described above that there are advantages in sequential storage. Because of this, the BMM in a

preferred embodiment stores packets into memory 102 in a manner to create a small number of large available regions, rather than a larger number of smaller available regions.

5   3. BMM 302 notifies processor 100 on path 310 when enough of the packet is stored, so that the processor can begin to perform the desired processing. An identifier for this structure is created and provided to the processor. The identifier communicates at a minimum the starting address of the packet in memory, and in some cases includes additional information.

10

4. BMM updates map 304 for all changes in the topology of the memory. This updating can be done in any of several ways, such as periodically, or every time a unit in memory is changed.

15   5. When processing is complete on a packet the BMM has stored in memory 102, the processor notifies BMM 302, which then transfers the processed data back to device 106. This is for the particular example of a packet processing task. In some other embodiments data may be read out of memory 102 by MM 302 and sent to different devices, or even discarded.

20   In notifying the BMM of processed data, the processor used the data structure identifier previously sent by the BMM upon storage of the data in memory 102.

6. The BMM updates map 304 again, and every time it causes a change in

25   the state of memory 102. Specifically the BMM de-allocates the region or regions of memory previously allocated to the data structure and sets them as available for storage of other data structures, in this case packets.

It will be apparent to the skilled artisan that there may be many alterations in the embodiments described above without departing from the spirit and scope of the present invention. For example, a specific case of operations in a data packet router was illustrated. This is a single instance

5   eof a system wherein the invention may provide significant advantages. There are many other systems and processes that will benefit as well. Further, there are a number of ways BMM 302 may be implemented to perform the functionality described above, and there are many systems incorporating many different kinds of processors that might benefit.

10

Low Fragmentation Data Storage

In the following described examples memory management is accomplished in a dynamic multi-streaming processor know to the inventors

15   as XCaliber, which has been described in one or more of the documents incorporated in the cross-reference section above.

Fig. 4 is a simplified diagram of memory space managed by XCaliber according to an embodiment of the present invention. Shown in the diagram are sections of memory space of the XCaliber multi-streaming

20   processor that are hardware controlled, software controlled, and other types of memory not specifically described. In this example, a specific section is labeled Hardware Controlled. The memory space of this section is analogous to LPM 219 described with reference to Fig. 2 of S/N 09/737,375 or memory 102 described with reference to Fig. 3 of S/N 09/602,279. In

25   this example, only a specified section of the total available memory of XCaliber is designated as hardware-controlled.

Also indicated by directional arrows in this example are Packets In that are received at the processor from a network such as, for example, the well-known Internet network. Packets Out, similarly indicated in this

example by directional arrows, indicate data packets that have been processed by XCaliber and are being uploaded for routing to designated destinations either internal to the router or over a network or networks, which may include the Internet network, to other routing points.

5   The section of hardware-controlled memory illustrated herein is controlled by hardware that is provided according to a preferred embodiment of the present invention and enhanced to manage the memory according to a provided protocol. In an embodiment of this invention it is preferred that incoming data packets are stored into and read out of

10   hardware controlled memory so that the central processing unit (CPU) or other processing resources do not have to perform costly operations involved in storing and reading out the data.

Although it is not explicitly indicated in this example, but is further described below, the section of memory labeled as hardware-controlled

15   memory is divided into a plurality of manageable blocks. It is possible in an embodiment of this invention that software can control none, one, or more memory blocks and leave those blocks not controlled by software to control of the hardware algorithm. Configuration flags are provided for indicating assigned software control of any one or more of memory blocks. When

20   such a flag is set the hardware controller will not store any incoming data packets into the flagged block.

The protocol provided in embodiments of this invention, defined by a specific algorithm, determines if any incoming data packets fit into any hardware-controlled blocks of memory. If incoming data packets fit into

25   any of the hardware-controlled blocks, the hardware algorithm enables a computation to determine which blocks within the hardware-controlled memory will be selected that will accommodate incoming data packets.

The novel protocol of the present invention introduces a concept of virtual and atomic pages as data storage containers of the hardware-

controlled memory. In a preferred embodiment, Virtual pages comprise a number of atomic pages. A goal of the present invention is to be able to reduce fragmentation that typically occurs when queuing and de-queuing data packets.

Fig. 5 is a block-diagram illustrating an example of a virtual page according to an embodiment of the present invention. This example illustrates just one of a plurality of divided sections of the hardware-controlled memory described above with reference to Fig. 4.

In actual practice, the hardware-controlled portion of memory of Fig. 4 is divided into 4 blocks each having 64 Kb total memory space. Therefore, a total size of the hardware-controlled memory of Fig. 4 is 256 Kb. This should, however, not be construed as a limitation of the present invention, as there are a number of possible division schemes as well as possible differing amounts of provided on-board memory. In this example only a single block of 64Kb is represented for simplicity in description.

The 64KB block of this example comprises a plurality of atomic page divisions having 256 bytes of memory space each. Therefore, there are in this example, 256 atomic pages making up a single 64 Kb block and 1024 atomic pages defining the four 64 Kb divisions of the total hardware-controlled memory referred to in the example of Fig. 4 above.

Graphically represented to the right of the 64 Kb memory block in this example are columns representing some possible allocated sizes of virtual pages. For example, a 256-byte virtual page (VP) size may exist that comprises a single atomic page (1:1) thus providing 256 (0-255) VPs per 64Kb block. A 512-byte VP size may exist with each VP comprising 2 atomic pages (2:1) thus providing 128 (0-127) VPs per block. Similarly, reading further columns to the right, virtual pages may comprise 1Kb of memory (0 through 63 atomic pages), 2Kb of memory (0 through 31 atomic

pages) and so on, according to power of 2 increments, up to a single 64Kb VP comprising the entire 64 Kb block.

An enhanced hardware mechanism is provided and termed HAL by the inventor, and is subsequently referred to as HAL in this specification. HAL computes and maintains a flag for each virtual page within a controlled memory block in order to determine whether a virtual page has been allocated for data storage or not. The status, including size of all atomic pages is, of course, known to HAL to make computations regarding whether or not to store an incoming data packet in a particular space.

Fig. 6a is a block diagram illustrating a first part of a two-part process of storing data packets into hardware-controlled memory according to an embodiment of the present invention. In the two-part function, HAL makes a determination whether a particular incoming data packet fits into any of the blocks of the hardware-controlled memory. If a packet fits, it is determined how many atomic pages of memory space will be needed to store the data packet. After packet storage, the used space is marked as allocated for storage of the packet. When the packet is read out of queue, the formerly allocated space is then de-allocated or marked as free space for consideration in future storage.

As was previously described above, the hardware controlled memory is divided into a plurality blocks of a fixed size. In practice in this example, total memory controlled by hardware (HAL) is 256KB divided into 4 sub-blocks of 64KB each. As described with reference to Fig. 5 of this specification, each 64KB block is divided into smaller sub-blocks of atomic pages of 256 bytes each, which are used to construct virtual pages.

At left in Fig. 6a, there is illustrated 4 64Kb blocks of memory, which taken together equate to a total memory that is controlled by HAL. Each block, as previously described, may be hardware or software controlled. If a block is software controlled, it will be identified as such and

HAL will not utilize the block for packet storage. To the right of the 4 64Kb blocks, there is illustrated a state of indication for each block. For example, an area is set aside to indicate if a block is software controlled. If this area does not indicate by flag that it is software controlled, then an

5 allocated/de-allocated indication will be present. This is indicated by "Block 0 state through block 3 state. It is noted herein that computation by HAL is performed in parallel for each 64Kb block.

If it is determined by HAL that there is available hardware controlled memory and that one or more blocks have sufficient space that is

10 de-allocated, or does not hold data, then HAL determines if the packet fits into any of the eligible spaces. It is noted herein that the byte size of an incoming data packet is appended to the packet in this example in the first 2 bytes of the packet header. This is a convenience in a preferred embodiment, but is not limiting for purposes of the invention. In cases

15 where no size is appended, the hardware algorithm would simple receive all of the packet, and when it detects that the packet has been completely received, it would compute the size of the packet. In this way, (either way) HAL may efficiently determine eligible spaces to store the packet. In this scheme, data packets are stored consecutively and a goal is to have all of a

20 packet contained in a virtual page to reduce fragmentation.

Blocks are selected for storage based on eligibility, and in some cases priority. Information generated by HAL in case of packet fit includes a block #, the total number of atomic pages required to store the packet, and the location identifier of the first atomic page marking the beginning of the

25 stored data packet. Knowing the first atomic page and the size of the data packet stored is sufficient to simplify reading the packet out of the hardware-controlled memory, since packets are consecutively stored.

Whether hardware or software controlled, status of selected blocks of memory must be computed and maintained by whichever entity (hardware or software) is controlling selected blocks of memory.

To select appropriate blocks of memory, HAL must keep track of regions of memory wherein active data is stored and regions that are free and available for storage. Once data packets are sent out to another device or location, those areas of memory associated with that data are de-allocated and available to be used again for storage of new data packets to be stored into the memory. Once fit determination is made, the HAL records a block number, atomic pages needed for storage, and at least a first atomic page number as a data identifier, and provides that data identifier to the multi-streaming processor for management of data. If a fit determination cannot be made, the controlling entity (HAL or software) may have the option of storing data packets in external storage memory or dropping data packets.

Fig 6b is a block-diagram illustrating a virtual page allocation matrix of atomic pages needed to store data packet and the re-computation (allocated/de-allocated) of the state of virtual pages. Allocation of atomic pages is accomplished by fit determination logic established by the allocation matrix that is comprised of the state of each of all virtual pages per block. Computation is updated each time one or more atomic pages is allocated or de-allocated and is an input back into the determination logic.

The allocation matrix maintains computation of allocated and de-allocated virtual pages relative to 256 byte, 512 byte, 1Kb, and other power-of-two increments up to a 64Kb virtual page. Allocated and De-allocated state information is submitted as input to the fits determination logic for each packet as described above.

In this example, Block j has 0-255 atomic pages representing the smallest increment of 256 bytes. The power-of-two increments of construction are 256B virtual pages, 512 B virtual pages, 1 KB virtual

pages, up to a 64 KB virtual page. The instant mapping scheme selectable by power of two increments is a programmable feature that may be programmed on the fly during packet processing.

Motivation for changing the memory mapping scheme with regard to the size of virtual pages allocated for packet storage may, in one embodiment, be derived from statistical averaging of the size of data packets entering a data port over a given, and also programmable, period of time. A goal of the present invention is to continually select the best mapping scheme that enables data storage with minimum fragmentation. Therefore, the way that the local packet memory (hardware controlled) is mapped can vary according to need. The exact criteria for determining when to change the mapping scheme may be established using a threshold scheme that automatically triggers a dynamic re-mapping of hardware-controlled memory. Because of this flexibility, which is not available in prior art memory addressing schemes, fragmentation may be kept to a minimum. However, a trade-off exists in that using a power of 2 to define selectable VP sizes is not necessarily the best way to reduce fragmentation. It is utilized in a preferred embodiment because it greatly simplifies computation, requiring minimum circuitry, providing for a smaller and faster chip implementation.

The primary factors of concern in this specification are an Allocation Matrix, a Fits Vector, and an Index Vector. These primary factors are defined as follows:

- AllocationMatrix[VPSize][VPIndex]: indicates whether virtual page number VPIndex of size VPSize is already allocated or not.

- FitsVector[VPSize]: indicates whether a block has at least one non-allocated virtual page of size VPSize.

- IndexVector[VPSize]: if FitsVector[VPSize] is asserted, IndexVector[VPSize] contains an index of a non-allocated virtual page or pages of size VPSize.

5

Determination of VP size for any one of a plurality of hardware managed blocks is dynamically programmed and, in some cases, re-programmed according to learned results of operation as previously described above. A factor defining this ongoing determination is termed

10   EnableVector[VPSize].

The above-described factors always remain in an undefined state for any block managed by software instead of hardware.

A supporting algorithm expressed in software language for the fits

15   determination logic (for a data packet of size s bytes) is:

1) *Fits logic*: Check, for each of the blocks, whether the data packet fits in or not. If it fits, remember the virtual page size and the number of the first virtual page of that size.

20
$$For\ All\ Block\ j\ \underline{Do}\ (can\ be\ done\ in\ parallel):$$
$$Fits[j] = (s <= VPSize)\ AND\ FitsVector[VPSize]\ AND$$
$$Not\ SoftwareOwned$$
$$where\ VPSize\ is\ the\ smallest\ possible\ page\ size.$$
$$\underline{If}\ (Fits[j])$$

25
$$VPIndex[j] = IndexVector[VPSize]$$
$$MinVPS[j] = VPSize$$
$$\underline{Else}$$
$$MinVPS[j] = <Infinit>$$

30   2) *Block selection*: The blocks with the smallest virtual page (enabled or not) that is able to fit the data packet in are candidates. The block with the smallest enabled virtual page is selected.

$$\underline{If}\ Fits[j] = FALSE\ for\ all\ j\ \underline{Then}$$
35
$$<Packet\ does\ not\ fit\ in\ hardware-controlled\ memory>$$
$$\underline{Else}$$
$$C = set\ of\ blocks\ with\ smallest\ MinVPS\ AND\ Fits[MinVPS]$$
$$B = block\#\ in\ C\ with\ the\ smallest\ enabled\ virtual\ page$$

(if more than one exists, pick the smallest block number)

> If one or more blocks in C have virtual pages enabled <u>Then</u>
> Index = VPIndex[B]
> VPSize = MinVPS[B]
> NumAPs = ceil(S/256)
> packetPage = (B*64KB + Index*VPSize) >> 8
>
> <u>Else</u>
> <Packet does not fit in hardware-controller memory>

A packetPage is an atomic page number of the first atomic page that a data packet will occupy in hardware-controlled memory. The packetPage is offset within hardware-controlled memory and can be used to quickly identify and access all data of a packet stored consecutively after that page. The total number of atomic pages (NumAPs) needed to store a data packet is calculated and allocated. Data packet size is determined by examining the first 2 bytes of the packet header as previously described. Allocation of atomic pages for a selected block (j) is determined as follows:

- The allocation status of atomic pages in AllocationMatrix[Apsize][j..k], j being the first atomic page and k the last one (k-j+1 = NumAPs), are set to be allocated.

- The allocation status of virtual pages in AllocationMatrix[r][s] is updated following the mesh structure shown in Fig. 6b: a 2^{k+1}-byte virtual page is allocated if any of the two 2^k-byte virtual pages that it is composed of is allocated.

Figs. 7a through 8d are block diagrams illustrating allocation of atomic (and virtual) pages by HAL. The collective diagrams numbering 8 in total are associated in an ongoing sequence of page allocation and packet storage. The 8 diagrams are further associated in sets of two memory blocks each, for example, Figs 7a and 7b representing a first sequence utilizing 2 memory Blocks 0 and 1. In actual practice, there are 4 memory

blocks within hardware-controlled memory. The inventor illustrates 2 Blocks 0 and 1, each comprising 2 KB of memory for purpose of simplifying explanation.

Referring now to Fig. 7a, assume that Block 0 is hardware controlled, empty of data, and selected for packet storage. The size of a packet for storage is 256 bytes as is indicated above the block. Options for virtual memory allocation in variable sized virtual pages are displayed in columns to the right of Block 0 in increments of powers of 2. The smallest size page is an atomic page of 256 bytes. Therefore in Block 0 there are 8 atomic page divisions 0-7 adding up to 2 KB (total memory). In the first column labeled 256-byte Virtual Page, there is one page available (0-7) for each atomic division 0-7 because they are of the same size. In the next column labeled 512-byte Virtual Page, there are only 4 available virtual pages (0-3) representing total memory because of the power of 2 rule. The remaining columns labeled 1 KB Virtual Page and 2 KB Virtual Page (VP) are presented accordingly using the power of 2 rule.

Immediately below Block 0 is a columned table representing values of three Vectors described previously in this specification. These are, reading from top to bottom, Fits Vector, Index Vector, and Enable Vector. The values presented in the table are associated with the Virtual Page columns. In this example, atomic division 7 is crosshatched indicating current cycle VP allocation of a 256-byte packet. Indication of the VP allocation by cross-hatching is extended across the presented columns in each VP Size category. The cross-hatching in this example indicates that the corresponding atomic page is allocated. The virtual page that contains this atomic page is then not available.

HAL computes and selects the most optimum storage space for the packet based on determined and chosen values represented in the Vector table for each column. The Enable Vector is a preprogrammed constant

programmed for each power of 2 columns. The values of yes (Y) or no (N) represented for each column indicate whether or not the function of looking for an available Virtual Page in that column is enabled or not. The specific determination of enabling or disabling consideration of a specific size

5   Virtual Page during a computation cycle depends on outside considerations such as knowledge of average size packets arriving at a port over a given period of time, and any desire to reserve certain size Virtual Pages in a given Block for storage of a specified size or size range of data packets. The Enable Vector is a programmable optimization tool to enable optimum

10   data storage with even less fragmentation.

The Fits Vector is a determination of whether a packet will fit into an available Virtual Page as determined by known size of the packet, and the Index Vector is a pointer to a next available Virtual Page in each size column for fitting a packet. While the Fits Vector is result-oriented

15   (computed result), the Index Vector is selectable in case there is a plurality of Index slots empty of data and available for packet storage. For optimum data storage the last available VP that fits a packet is chosen for storage. It could also be the first available. Either way will work, as long as it is either the last available or the first available.

20   In this example, it is determined that for selected Block 0, a packet of the size of 256-bytes will fit in a 256-byte virtual page (indicated by cross hatching). In the event of storage of the packet in a 256-byte virtual page, an Index Vector of 6 (or the next 256-byte slot) is flagged for the next available "page" in memory for a next 256-byte packet. This represents the

25   most optimum storage use through consecutive storage and no fragmentation, using the scheme of power-of-two virtual pages and fixed size of atomic pages. The packet will also fit in a 512-byte virtual page, a 1 KB virtual page, and in a 2 KB virtual page. A tabled Y for Enable Vector

indication is not required in the case of a 2 KB virtual page as that page
represents the total memory selected.

If the 256-byte packet is stored in a 512 Virtual Page it would
occupy a block in that column representing atomic divisions 6 and 7 within

5   Block 0 according to power of 2. In this case the Vectors read Y=fits, 2
(chosen as pointer for next available 512-byte Virtual Page), and Y=enabled
for consideration. If the packets coming in average between 256 and 512
bytes, it is logical to reserve 512 byte pages as indicated by Enable Vector
value of Y for that column. It is reminded that there are three other blocks

10   in actual practice that can be hardware controlled.

Referring now to Fig. 7b, Block 1 represents the other hardware-
controlled memory block of this example. The absence of any activity
designated by cross-hatching simply means that Block 1 has not been
selected for packet storage in the first cycle.

15   Referring now to Fig. 7c, the activity represented in Fig. 7a is
present in Block 0 as double crosshatched blocks for the packet of 265-
bytes. For a next packet of 512-bytes in the next computation cycle, Block
0 in the column 512-bytes has atomic pages 4 and 5 allocated for receiving
the 512-byte packet. This allocation resulted from the previous index vector

20   of 2 represented with respect to Fig. 7a. In this sequence, only the index
vector value of 1 in the 512-byte column has changed indicating that block
as the next available 512-byte VP for a next packet of that size or smaller.
Referring now to Fig. 7d, an absence of cross-hatching indicates that Block
1 was not selected for packet storage in the current cycle.

25   Referring now to Fig. 8a, the sequence now must deal with fits
determination and allocation for a 1 KB data packet as is indicated above
Block 0. In this example, the previous activity described with reference to
Figs. 7a (256-byte) and 7c (512-byte) is illustrated herein as double
crosshatched blocks indicating past allocation and current ineligibility for

consideration in this current cycle. It is also noted that neither column (1 KB) nor column (2 KB) is enabled. Even though a 1 KB block may fit in the open VP in the 1 KB column, Block selection is deferred to Block 1 illustrated with reference to Fig. 8b. That is to say that Block 0 represented in Fig. 8a is not selected for storage of the 1 KB packet.

Referring now to Fig. 8b, Fits Vector is positive (Y) for all size columns. Atomic divisions 4-7 are allotted for storage of the 1 KB packet in the current cycle as indicated by crosshatching. Index Vector 3 represented in the 256-byte VP column indicates the next available storage index (VP) in the next cycle. It is noted herein that Enable Vector values are positive in the 1 KB and 2 KB columns. In the next cycle, there will be available 4 256-byte VPs (Index Vector 3), 2 512-byte VPs (Index Vector 1), and 1 KB VP (Index Vector 0), available for consideration for storage of a next packet. It is noted that VP 2-KB is not considered in the algorithm for a next cycle be cause it has been allotted.

Figs. 8c and 8d illustrate further operations involving packets of 512 bytes, and can be understood in light of the above descriptions.

Fig. 9 is an illustration of how memory space is better utilized by consecutive storage according to an embodiment of the present invention. This example illustrated two scenarios, A and B, wherein two 256-byte data packets are stored in a block. In SCENARIO A, a 256-byte virtual page is randomly chosen, whereas in SCENARIO B, the largest index vector is always chosen. As can be seen, the block in SCENARIO A only allows two 512-byte virtual pages to be considered at a next round whereas the block in SCENARIO B allows three VPs. Both, however, allow the same number of 256-byte data packets since this is the smallest allocation unit. The same optimization may be obtained by choosing the smallest virtual page index number all the time.

It is noted herein that assignment of Virtual Pages as explained in the examples of Figs. 7a through Fig. 9 is performed in parallel for all memory blocks of hardware-controlled memory that are not flagged for software control. It will be apparent to one skilled in the art that the embodiments of the invention described in this specification are exemplary, and may vary in a multitude of ways without departing from the spirit and scope of the present invention. It will also be apparent to one skilled in the art that many alternatives, variations, and configurations are possible and the scope of the invention is limited only by the claims that follow.